SFML - Događaji, slike i strukturiranje koda

Objektno programiranje - 5. vježbe

Marko Živković

Prirodoslovno-matematički fakultet,
Sveučilište u Zagrebu

10. travnja 2026. godine

Klasa `sf::Event`

- Samo funkcije `pollEvent` i `waitEvent` daju valjane događaje.

Tipična petlja za događaje:

```
sf::Event event;
while (prozor.pollEvent(event)) {
    switch (event.type) {
        case sf::Event::Closed:
            prozor.close();
            break;
        case sf::Event::KeyPressed:
            ...
            break;
        default:  //ako nas ne zanimaju ostali
            break;
        }
}
```

Događaj promjene veličine prozora

- pri promjeni veličine prozora (ili korisnik ručno ili u programu pozivom `prozor.setSize(...)`
- omogućuje promjenu postavki renderiranja nakon događaja

Primjer. Ispis dimenzija prozora pri promjeni veličine prozora:

```
while(prozor.isOpen()) {
  sf::Event event;
  while (prozor.pollEvent(event)) {
      switch (event.type) {
          case sf::Event::Closed:
              prozor.close();
              break;
          case sf::Event::Resized:
              cout << "(" << prozor.getSize().x << ","
                   << prozor.getSize().y << ")" << endl;
              break;
      }
  }
}
```

Događaji dobivanja i gubljenja fokusa

- promjena fokusa događa se kad promijeni trenutno aktivni prozor
- prozor koji nije u fokusu ne dobiva ulaz s tipkovnice
- primjer upotrebe: pauziranje igre kad prozor nije u fokusu

Primjer.

```
...
case sf::Event::GainedFocus:
    cout << "Dobio fokus!" << endl;
    break;
case sf::Event::LostFocus:
    cout << "Izgubio fokus!" << endl;
    break;
...
```

Ulazak miša u prozor i izlazak miša iz prozora

- `sf::Event::MouseEntered` - prilikom ulaska pokazivača miša u prozor
- `sf::Event::MouseLeft` - prilikom izlaska pokazivača miša iz prozor

Primjer.

```
...
case sf::Event::MouseEntered:
    cout << "Usao u prozor!" << endl;
    break;
case sf::Event::MouseLeft:
    cout << "Napustio prozor!" << endl;
    break;
...
```

Pomak miša

- **`sf::Event::MouseMoved`** - pri pomicanju miša **unutar prozora** (ne računa se naslovna traka niti rub)
- radi čak i ako prozor nije u fokusu
- dobivanje koordinata pokazivača miša (unutar prozora!):

(**`mouseMove.x`**, **`mouseMove.y`**)

Primjer.

```
...
case sf::Event::MouseMoved:
    cout << "(" << event.mouseMove.x << ","
         << event.mouseMove.y << ")" << endl;
    break;
...
```

Pritisak i otpuštanje tipke miša

- `sf::Event::MouseButtonPressed` - pritisak tipke miša
- `sf::Event::MouseButtonReleased` - otpuštanje tipke miša
- koordinate pokazivača dobivamo pomoću `mouseButton`
- tipke miša koje SFML podržava: `Left` (lijevo), `Right` (desno), `Middle` (kotačić), `XButton1`, `XButton2` (tipke sa strane)

Primjer.

```
...
case sf::Event::MouseButtonPressed:
    if(event.mouseButton.button == sf::Mouse::Left)
        cout << "Lijevi klik na ("
             << event.mouseButton.x << ","
             << event.mouseButton.y << ")" << endl;
    break;
...
```

Napomena. Ostali događaji (poput `MouseWheelScrolled`): link

Pritisak i otpuštanje tipke na tipkovnici

- događaji tipa **`KeyPressed`** i **`KeyReleased`**
- to koristimo za **reakciju** na pritisak tipke (npr. pomicanje lika u igri), a ne `TextEntered` događaje (za unos teksta - njih ćemo raditi kasnije)

Primjer.

```
...
case sf::Event::KeyPressed:
    cout << "Tipka!" << endl;
    break;
...
```

- pritisnite i držite neku tipku: generira se više `KeyPressed` događaja, s *defaultnim* **kašnjenjem** (isto kao pri pisanju u neki dokument)
- za onemogućavanje ponavljanja `KeyPressed` događaja pri držanju pritisnute tipke treba na prozoru izvršiti:

```
prozor.setKeyRepeatEnabled(false)
```

Kako provjeriti koja tipka je pritisnuta

- `key.code` daje kod pritisnute tipke
- kodovi - `sf::Keyboard::`
  - A, B, C, ..., X, Y, Z, Num0, Num1, ..., Num9
  - LControl, LShift, LAlt, LSystem, RControl, RShift, RAlt, RSystem
  - Escape, Menu, Pause
  - LBracket ([), RBracket (]), Semicolon (;), Comma (,), Period (.), Quote (‘), Slash (/), Backslash (\), Tilde (~), Equal (=), Hyphen (-)
  - Space, Enter, Backspace, Tab, PageUp, PageDown, End, Home, Insert, Delete
  - Add (+), Subtract (-), Multiply (*), Divide (/)
  - Left ($\leftarrow$), Right ($\rightarrow$), Up ($\uparrow$), Down ($\downarrow$)
  - Numpad0, Numpad1, ..., Numpad9
  - F1, F2, ..., F15
- sljedeći kodovi su zastarjeli (u zagradama je navedeno čime su zamijenjeni):
  - Dash (Hyphen)
  - BackSpace (Backspace)
  - BackSlash (Backslash)
  - SemiColon (Semicolon)
  - Return (Enter)

Još malo o tipkama

- funkcija za provjeru je li određena tipka trenutno pritisnuta:

```
static bool sf::Keyboard::isKeyPressed(Key key)
```

Napomena.

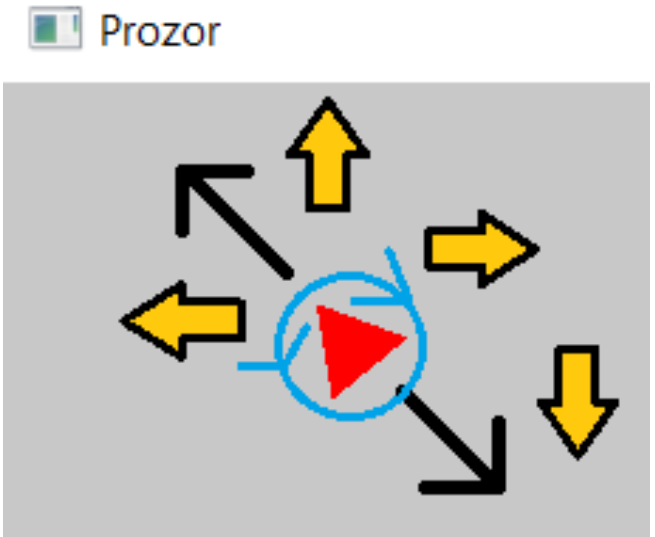
- neke tipke imaju posebno značenje (ovisno o operacijskom sustavu) pa to može dovesti do neočekivanog ponašanja
- primjerice, na Windowsu i Visual studiu tipka F12 pokreće *debugger*

Primjer

```
...
case sf::Event::KeyPressed:
    if(event.key.code == sf::Keyboard::Up)
        cout << "Tipka gore!" << endl;
    break;
...
```

Zadatak

Zadatak 1. Nacrtati crveni jednakostranični trokut (radijus opisane kružnice je 20 piksela) koji se početno nalazi na sredini 640×480 sivog prozora te koji se može pomicati pritiskanjem strelica na tipkovnici (kao na slici ispod: strelica gore je naprijed, strelica dolje je natrag, strelica lijevo je rotacija u smjeru obrnutom od smjera kazaljke na satu, a strelica desno je rotacija u smjeru kazaljke na satu).



Rješenje

```
sf::Vector2f pocetna_brzina(0.f, -200.f); //gore
float brzina_okreta(200.f);
int main() {
    sf::RenderWindow prozor(sf::VideoMode(640,480),
                                    "Prozor");
    float radijus = 20.f;
    sf::CircleShape t(radijus,3);
    t.setFillColor(sf::Color::Red);
    t.setOrigin(radijus, radijus);
    t.setPosition(prozor.getSize().x / 2.f,
                  prozor.getSize().y / 2.f);
    sf::Vector2f brzina(pocetna_brzina);
    float kut = 0;
    sf::Clock clock;
    while (prozor.isOpen()) {
        sf::Event event;
        ...sljedeći slajd...
    }
}
```

Nastavak rješenja (unutar `while` petlje)

```
while (prozor.pollEvent(event))
    if(event.type == sf::Event::Closed)
        prozor.close();
float deltaTime = clock.restart().asSeconds();
if(sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
    t.move(brzina * deltaTime);
if(sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
    t.move(-brzina * deltaTime);
if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left)){
    t.setRotation(kut -= brzina_okreta *
deltaTime);
    update_brzine(brzina, t.getRotation());
}
if(sf::Keyboard::isKeyPressed(sf::Keyboard::Right)){
    t.setRotation(kut += brzina_okreta *
deltaTime);
    update_brzine(brzina, t.getRotation());
}
```

Nastavak unutar `while` petlje

```
prozor.clear(sf::Color(200, 200, 200, 255));
prozor.draw(t);
prozor.display();
```

Pomoćna funkcija

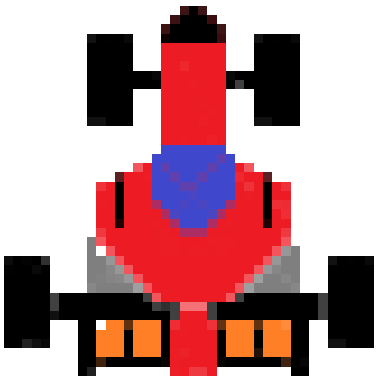
```
void update_brzine(sf::Vector2f& brzina, float
kut){
    float rad = kut / 180 * 3.14;
    brzina.x = cos(rad) * pocetna_brzina.x
             - sin(rad) * pocetna_brzina.y;
    brzina.y = sin(rad) * pocetna_brzina.x
             + cos(rad) * pocetna_brzina.y;
}
```

- ovo zahtijeva `#include<cmath>`
- `kut` je u stupnjevima, a za funkcije `sin` i `cos` trebamo u radijanima - zato smo prvo pretvorili stupnjeve u radijane

Upotreba slike

Zadatak 2. Promijeniti trokut iz prošlog zadatka u formulu. Datoteka "formula.png" može se preuzeti na web-stranici kolegija.

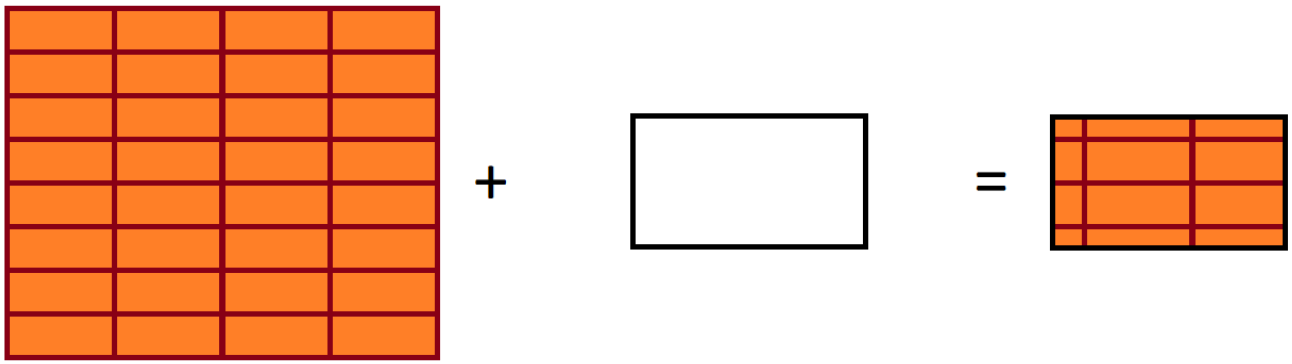
- datoteka je nastala u programu *Paint*, te joj je dodatnim programom pozadina stavljena na transparentnu
- dimenzije slike su 40 $\times$ 40 piksela



Spriteovi i teksture

- **tekstura** = slika (nazivamo ju tekstura jer se preslikava na 2D objekt)
- ***sprite*** = pravokutnik s teksturom

Ilustracija:



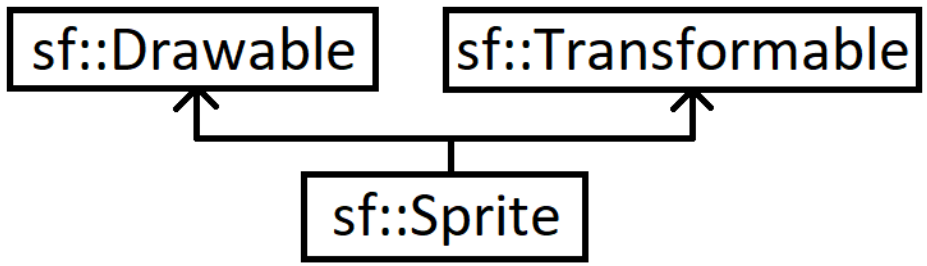
Klase za *sprite* i za teksturu

Tekstura

- slika koja se nalazi na grafičkoj kartici i može se koristiti za crtanje
- `sf::Texture` - sprema piksele koji se mogu crtati (npr. pomoću *spritea*)
- no, to nije objekt koji npr. pomičemo tijekom igre

Sprite

- slika koja se može koristiti kao 2D objekt, koji ima koordinate, boju i može se pomicati, uništiti ili stvoriti tijekom igre
- klasa `sf::Sprite` - dijagram nasljeđivanja:



Tekstura

- s obzirom da je jedina uloga teksture učitavanje i preslikavanje na grafički objekt, većina funkcija koje ima su za učitavanje i ažuriranje teksture

U kodu na početku `main` funkcije dodajmo:

```
sf::Texture texture;
texture.loadFromFile("formula.png");
```

Napomena. Datoteka `"formula.png"` mora se nalaziti **u istoj mapi kao i projekt** (tamo gdje se nalazi `.vcxproj` datoteka). U protivnom se na standardni izlaz ispiše poruka:

```
Failed to load image "formula.png".
Reason: Unable to open file
```

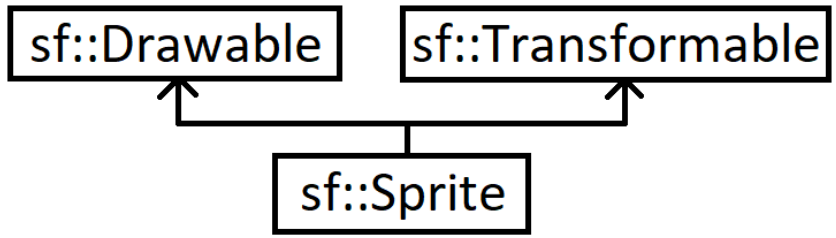
Upotreba klase `sf::Sprite`

- u kodu umjesto `sf::CircleShape t...` i `t.setFillC...`:

```
sf::Sprite t;
t.setTexture(texture);
```

Uočimo da ostalo u kodu ne trebamo mijenjati:

- crtanje je i dalje ovako: `prozor.draw(t);`
- podsjetnik na dijagram nasljeđivanja:



- kako klasa `sf::Sprite` nasljeđuje klasu `sf::Transformable`, onda također ima metode za transformacije:
  - `setOrigin`
  - `setPosition`
  - `move`
  - `setRotation`

Zašto nam treba strukturiranje

- Iz prethodnog koda mogli bismo napraviti neku igru - možemo dodati:
  - provjeru kolizije formule s rubom ekrana,
  - stazu za utrke,
  - protivničke formule,
  - prepreke na stazi,
  - bodovanje,
  - ...
- No, porastom količine koda on postaje **težak za održavanje**.
- Zato strukturiramo kod tako da stvorimo posebnu klasu za pojedini element programa.

Klasa `Igra`

- napraviti novu datoteku **`Igra.h`** u projektu
- Htjeli bismo sljedeću `main` funkciju

```
Igra igra;
while(!igra.gotovo()){
    igra.update();
    igra.renderiraj();
}
```

Klasa Igra

```
#include <SFML/Graphics.hpp>
#include <cmath>
class Igra {
    public:
        Igra();
        ~Igra();
        bool gotovo() {
            return !prozor.isOpen()
        }
    private:
        sf::RenderWindow prozor;
};
Igra::Igra() : prozor(sf::VideoMode(640, 480),
"Prozor"){}
Igra::~Igra() {}
```

Što sve trebamo pamtiti o formuli

```
class Igra {
    ...
    private:
        ...
        sf::Texture texture;
        sf::Sprite t;
        const sf::Vector2f pocetna_brzina;
        sf::Vector2f brzina;
        const float brzina_okreta;
        float kut;
        sf::Clock clock;
};
```

Inicijalizacija potrebnih dijelova u konstruktoru

```
Igra::Igra() :
      prozor(sf::VideoMode(640, 480), "Prozor"),
      pocetna_brzina(0.f, -200.f),
      brzina_okreta(200.f) {
   texture.loadFromFile("formula.png");
   t.setTexture(texture);
   t.setOrigin(t.getLocalBounds().width / 2,
t.getLocalBounds().height / 2);
   t.setPosition(prozor.getSize().x / 2.f,
prozor.getSize().y / 2);
   brzina = pocetna_brzina;
   kut = 0;
}
```

Funkcija za ažuriranje

Funkcija `update_brzine` može postati privatna funkcija klase `Igra`.

```
class Igra {
    private:
        void update_brzine();
        ...
};

void Igra::update_brzine() {
...
}
```

Funkcija za ažuriranje

```
void Igra::update() {
sf::Event event;
while (prozor.pollEvent(event))
    if(event.type == sf::Event::Closed)
        prozor.close();
float deltaTime = clock.restart().asSeconds();
if(sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
    t.move(brzina * deltaTime);
if(sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
    t.move(-brzina * deltaTime);
if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left)){
    t.setRotation(kut -= brzina_okreta *
deltaTime);
    update_brzine(brzina, t.getRotation());
}
if(sf::Keyboard::isKeyPressed(sf::Keyboard::Right)){
    t.setRotation(kut += brzina_okreta *
deltaTime);
```

Funkcija za renderiranje

```
class Igra {
    public:
        ...
        void renderiraj();
        ...
};

void Igra::renderiraj() {
    prozor.clear(sf::Color(200, 200, 200, 255));
    prozor.draw(t);
    prozor.display();
}
```

Glavni program (funkcija `main`)

```
#include <iostream>
#include <SFML/Graphics.hpp>
#include "Igra.h"
using namespace std;

int main() {
    Igra igra;
    while(!igra.gotovo()){
        igra.update();
        igra.renderiraj();
    }
}
```

Zadatak: Funkcija za ažuriranje

Zadatak 3. Razdvojite funkciju za ažuriranje na opći dio koji je potreban kod svake igre (zatvaranje prozora), i specijalni dio za pomicanje u ovoj igri. To može omogućiti baznu klasu za više igara u kojoj će biti funkcija s općim dijelom.

Rješenje

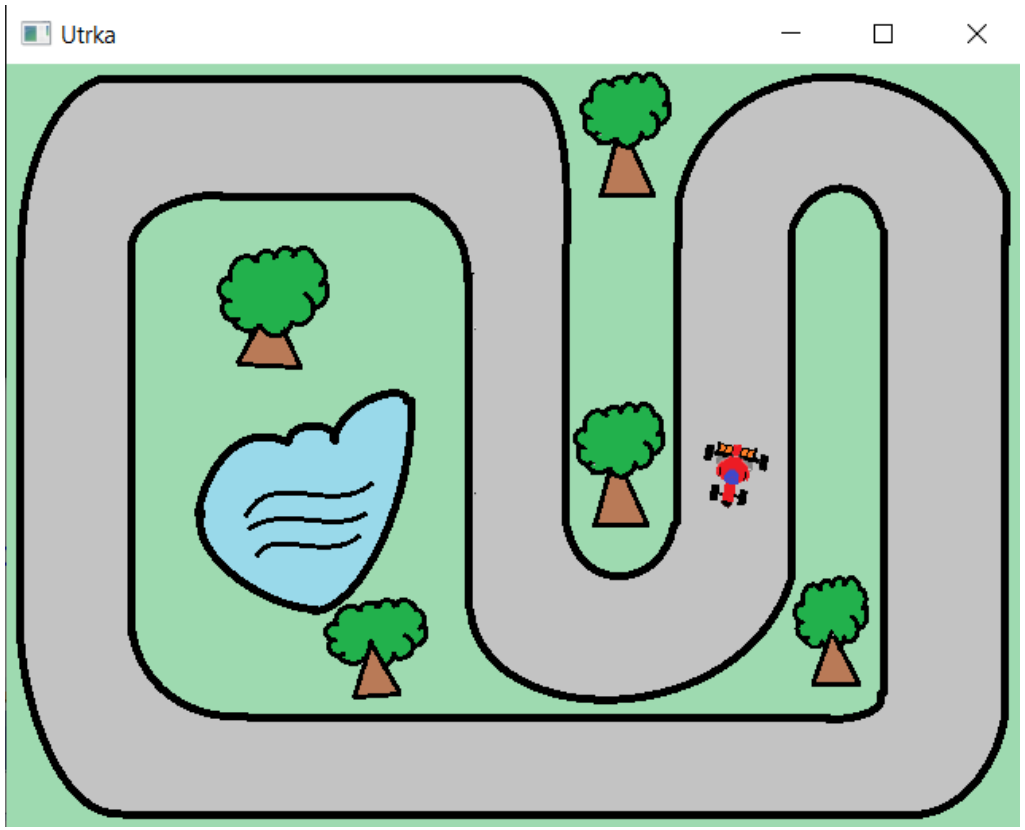
```
class Igra {
    protected:
        ...
        virtual void innerUpdate(sf::Event& event);
        ...
};

void Igra::update() {
    sf::Event event;
    while (prozor.pollEvent(event))
        if(event.type == sf::Event::Closed)
            prozor.close();
    innerUpdate(event);
}
```

innerUpdate

```
void Igra::innerUpdate(sf::Event& event) {
float deltaTime = clock.restart().asSeconds();
if(sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
    t.move(brzina * deltaTime);
if(sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
    t.move(-brzina * deltaTime);
if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left)){
    t.setRotation(kut -= brzina_okreta *
deltaTime);
    update_brzine(brzina, t.getRotation());
}
if(sf::Keyboard::isKeyPressed(sf::Keyboard::Right)){
    t.setRotation(kut += brzina_okreta *
deltaTime);
    update_brzine(brzina, t.getRotation());
}
```

Zadatak

Zadatak 4. Dodati u pozadinu mapu (npr. može se iskoristiti datoteka `mapa.png` koja se može preuzeti na web-stranici kolegija).



Rješenje

```
class Igra {
    private:
        ...
        sf::Texture textureMapa;
        sf::Sprite spriteMapa;
        ...
};
void Igra::Igra() ... {
    ...
    textureMapa.loadFromFile("mapa.png");
    spriteMapa.setTexture(textureMapa);
    ...
}
void Igra::renderiraj() ... {
    ...
    prozor.draw(spriteMapa);
    ...
}
```